Breadth First Search: Shortest Reach

Link submit: https://www.hackerrank.com/challenges/bfsshortreach

Solution:

C++	http://ideone.com/snxZGi
Java	http://ideone.com/UvpQF7
Python	http://ideone.com/eFTvop

Tóm tắt đề:

Cho một đồ thị vô hướng có n đỉnh được đánh số từ 1 đến n. Khoảng cách giữa 2 đỉnh luôn có độ dài là 6.

Bạn có q câu truy vấn, mỗi câu truy vấn tìm chi phí đường đi ngắn nhất từ đỉnh s cho trước đến tất cả các đỉnh còn lại. In ra -1 nếu không có đường đi giữa 2 đỉnh.

Input:

Dòng đầu tiên chứa số lượng câu truy vấn *q* (1 ≤ q ≤ 10), mỗi câu truy vấn có định dạng như sau:

- Dòng đầu tiên chứa hai số *n*, *m* (2 ≤ n ≤ 1000, 1 ≤ m ≤ n * (n - 1) / 2) với n là số nút, m là số cạnh.
- m dòng tiếp theo, mỗi dòng gồm hai số *u*, *v* (1 ≤ u, v ≤ n) tức hai đỉnh có kết nối với nhau.
- Dòng cuối cùng chứa đỉnh *s* là đỉnh cần tìm đường đi ngắn nhất từ đỉnh s này đến tất cả các đỉnh còn lại (1 ≤ s ≤ n).

Output:

Với mỗi truy vấn, in n – 1 số trên một dòng, mỗi số cách nhau dấu cách đại diện cho chi phí đi từ đỉnh s đến các đỉnh (1, 2, ..., n không bao gồm s) trong đồ thị.

Ví dụ:

2 4 2 1 2 1 3 1 3 1	6 6 -1 -1 6

2 3 2	

Giải thích ví dụ:

Có 2 truy vấn:

Truy vấn 1: có 4 đỉnh và 2 cạnh. Đỉnh 1 nối với đỉnh 2, đỉnh 1 nối với đỉnh 3. Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại.

- Đỉnh 1 → 2: 1 * 6 = 6.
- Đỉnh 1 → 3: 1 * 6 = 6.
- Đỉnh 1 → 4: -1 (Do không có đường đi).

Truy vấn 2: có 3 đỉnh và 1 cạnh. Đỉnh 2 nối với đỉnh 3. Tìm đường đi ngắn nhất từ đỉnh 2 đến tất cả các đỉnh còn lại.

- Đỉnh 2 → 1: -1 (Do không có đường đi).
- Đỉnh 2 → 3: 1 * 6 = 6.

Hướng dẫn giải:

Bài này áp dụng BFS cơ bản, đọc vào số lượng đỉnh của đồ thị và danh sách cạnh. Sau đó chạy BFS bắt đầu từ điểm s. Viết một hàm đếm các cạnh đi qua từ s đến các đỉnh khác. Lấy kết quả đếm nhân 6 để ra kết quả cần tìm.

Tuy nhiên ta có thể biến tấu thuật toán BFS đôi chút để có được kết quả ngay lúc duyệt đồ thị.

Gọi dist[v] lưu khoảng cách ngắn nhất từ đỉnh s đến đỉnh v. Với đỉnh v có được thông qua duyệt các đỉnh kề của đỉnh u. Như vậy dễ dàng nhận thấy dist[v] = dist[u] + 1, nghĩa là từ s đến v ta mất một quãng đường bằng khoảng cách từ s đến u và từ u đến v (chưa tính trọng số 6 của mỗi cạnh).

Cuối cùng cho vòng lặp i duyệt lại toàn bộ các đỉnh trong đồ thị. Nếu đỉnh đó chưa được viếng thăm thì in -1, ngược lại in dist[i] * 6 với trọng số của mỗi cạnh là 6.

Độ phức tạp: **O(T * (V + E))** với T là số lượng test, V là số lượng đỉnh trong đồ thị và E là số lượng cạnh trong đồ thị.